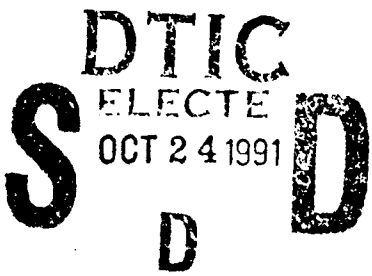


Mathematical Basis for a System to Manage Automated Protocol Analysis

TR91-024
April, 1991

Richard M. Hawkes

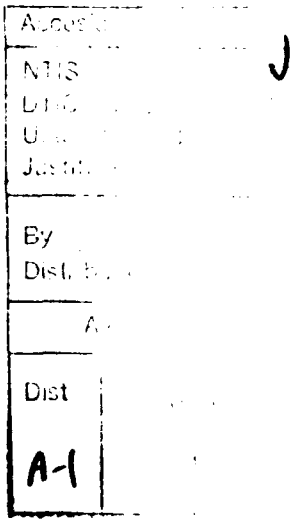
The University of North Carolina at Chapel Hill
Department of Computer Science
CB#3175, Sitterson Hall
Chapel Hill, NC 27599-3175
919-962-1792
jbs@cs.unc.edu

A TextLab Report

This research was supported by NSF (Grants # IRI-8519517 and IRI-8817305), the Army Research Institute (Contract # MDA903-86-C-345) and by the Office of Naval Research (Contract # N00014-86-K-00680).

Abstract

The TextLab Research Group has developed a number of tools and techniques for automatically recording users' interactions with computer systems in machine-readable form, for replaying sessions, for analyzing protocol data using cognitive grammars, for filtering analyzed data and interfacing with statistical packages, and for displaying results in visual forms that facilitate interpretation. The methodology they are developing encourages the collection of large numbers of protocols that must be stored, retrieved, divided into meaningful groups, etc., before they can be analyzed. Thus, *managing* protocol data becomes increasingly important. Our long-term goal is to develop an integrated environment from which to control and monitor all stages of the process; the goal of this paper is to provide a conceptual foundation for that system. It discusses issues concerned with sorting and selecting protocols according to associated attributes and with criteria for the proper application of particular statistical or other analytic functions to particular forms of protocols or data derived from them. The mathematical model presented is general and can be applied to other applications in which matching analytic program requirements with data type or organization is important.



Introduction

During the past six years, the TextLab Research Group has developed a number of tools and techniques for automatically recording users' interactions with computer systems in machine-readable form. They have also developed tools for replaying sessions, for analyzing protocol data using cognitive grammars, for filtering analyzed data and interfacing with statistical packages, and for displaying results in visual forms that facilitate interpretation. While the human interpreter is an integral and indispensable part of this process, these tools automate many of the protocol analysis steps, making it practical to analyze behaviors of large numbers of subjects over extended periods of time under both naturalistic and controlled conditions. Consequently, this methodology inevitably leads to the collection of large numbers of protocols that must be stored, retrieved, divided into meaningful groups, etc., before they can be analyzed. Thus, *managing* protocol data becomes increasingly important.

Our long-term goal is to develop an integrated environment that researchers can use to control and monitor all stages of the process. While exploring requirements and design criteria, we encountered several difficult, but interesting, issues concerned with sorting and selecting protocols according to various criteria and with applying statistical and other analytic functions to the selected protocols. This paper discusses these issues in more detail to provide a conceptual foundation for the system we plan to build.

This paper provides a mathematical specification of a data analysis system. The discussion may be of interest to anyone engaged in the high-level specification of data analysis systems. In general, the reader should find here ways in which data analysis can be formally modeled. The reader will see too how this model of data analysis is reflected in a prototype system. It may be that specific objects in the mathematical specification can be adapted or extended to solve problems in other types of data analysis.

The mathematical specification presented here represents the objects and tasks required for analyzing machine-recorded protocols. Protocols present two special difficulties during analysis. These data sets are often both massive and cryptic. The mathematical model described tries to answer these difficulties.

This model is not comprehensive; it is a framework to aid the work of system design. It is meant to demonstrate how a protocol analysis

system can be mathematically modeled as a step toward a prototype. This model can be extended or implemented in a number of ways, depending on specific goals. It forms the basis for the prototype described in the third section and could serve as a guide in extending the generality of that prototype.

This first section will discuss the problem of analyzing protocols. The second section will give a mathematical specification to model the analysis process. The third section suggests how this model of data analysis might lead to a software system providing a graphical and orderly approach to analyzing protocols.

What Is a Protocol?

This paper will use the word *protocol* to refer to a machine-generated record of a process. Such a record usually takes the form of a time-stamped list of events summarizing for later evaluation the course of the process.

For example, the analysis tool described in this paper was motivated by a need to handle sessions recorded by a writing support system known as WE (Writing Environment). WE records each user action in the process of organizing, writing, and rewriting a document. The resultant protocol is large and nearly useless without automatic processing. For the purposes of this paper, any such recorded list of discrete events may be considered to be a protocol.

These protocols typically have built into them two characteristics which make direct manipulation and analysis difficult. First, they are massive: since computers can record discrete events very rapidly, the size of such a recording is usually orders of magnitude larger than what might be produced by a human observer. Second, they are, for the human reader, cryptic, ordinarily being recorded in a notation chosen for ease of machine manipulation and information density.

Computer-assisted analysis seems a good way to deal with machine-generated protocols for two reasons:

First, though large, a protocol is easily packaged as an individual file. Thus, the manipulation of protocols can, in part, make use of the file system. Since computers normally have in place this system for handling large data objects, a system for the specialized manipulation of protocols already has a good foundation on which to build.

Second, the protocol can record its observations in a language suited specifically to machines. Automatic transformations can readily be made using this language. Results meaningful to the experimenter may be

produced after a series of automatic machine operations on the cryptic data set.

What Is Analysis?

The purpose of data analysis is to make explicit the implicit. Data analysis tries to find meaningful patterns where no specific meaning is known to exist. Analytical understanding is arrived at through the repetition of a two-part process: distinction and relation. In other words, the experimenter has first to isolate the variables of interest and then to see how they relate to each other. This model of analysis rests on the merits of long-use and long-usefulness, being suggested by Aristotle in his *Physics* and elaborated by Francis Bacon in his *Novum Organum*.

This process of distinguishing variables followed by relating them suggests a possible form for a graphical tool to assist in the task of analysis: a diverging tree, in which finer and finer distinctions are established, used as input to another, converging tree of increasingly general relationships (see Figure 1).

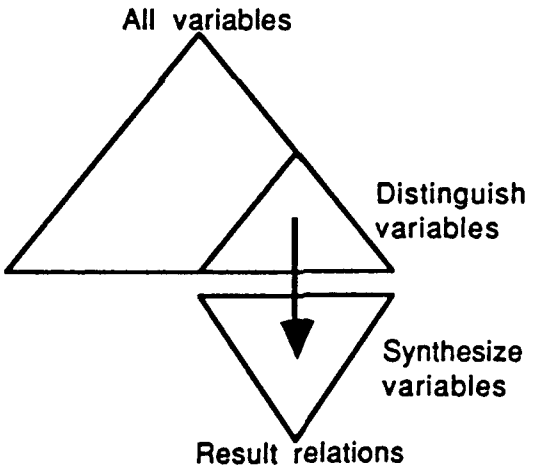


Figure 1:
A simple model of analysis

The analysis system described here follows this basic diamond form of top-down sorting followed by bottom-up synthesis. Iteration of such analysis then leads to the perception of meaningful patterns and an understanding of the nature of the things observed.

Design of a Protocol Analysis Tool

What characteristics should be reflected in the mathematical specification of a protocol analysis tool? Such a tool must manipulate many large data sets, allowing the experimenter to collect, categorize, and distinguish the data sets. Manipulation must be easy, encouraging exploration of many possible groupings. This tool should allow the protocols to be gathered into groups with similar traits for comparison among themselves and with other groups.

Also, this tool must provide for meaningful transformations of the data in order to discover implicit patterns and relations. This is not the execution of a prescribed algorithm or predefined solution. Rather, it is an interactive execution of transformations exploring likely paths, hunting for an unknown solution. Operators in this system should be dynamically typed, putting less of a demand on the user to fit data to operators. Operators should be flexible in the input they accept and should produce a predictable output so that the composition of operators is simple.

This tool must include not only capacities for manipulation and transformation, but also for iterations of these operations. The desired tool will provide a memory, a thread laid through the maze of all possible analytical paths. It should recall which protocols were used as input to which operators. Then a previous path may be altered by a single step, generating a new result while retaining the context of the simple change that caused it. This ability provides the retrospection needed in the usual course of data analysis.

The next section describes the mathematical constructs which seem likely pieces to model such a tool. These constructs provide for the ready manipulation of formally defined protocols. They also include operators which are able dynamically to test their fit to groupings of protocols. The third section of this paper will then show how these constructs can be used in a prototype system.

Mathematical Description of a Protocol Analysis System

The mathematics of this section is intended to serve as a formal specification for a system to analyze protocols. The need for such a specification became clear during a first attempt at programming a protocol analysis system. The implementation ran up against many limitations, and it became unclear how deeply design changes would have to reach in order to obtain a prototype supporting the tasks typical of protocol analysis. By rigorously modeling the objects and operations involved in such analysis, this mathematical specification provides one way in which the pieces needed for the system can be made to fit together logically. This formalism also allows anyone extending the system to determine whether a certain extension is possible and how it must be designed in order to conform to the existing system.

This section describes two basic mathematical objects that serve as abstractions for the manipulation and transformation of protocols. The first subsection will define a sort tree for the selection and grouping of protocols. The second subsection will describe the form of an operator which may be applied to these groupings.

Attribute-Value Trees

An attribute-value tree (AVtree) is similar to a full, balanced tree and is used for the selection and sorting of protocols. The protocols of an AVtree are formally represented as its data objects. Each node of an AVtree represents a particular set of data objects (protocols). Each level in the AVtree is identified with an attribute. Data objects are sorted at a particular level according to their values for the attribute of the level. Figure 2 illustrates how a set of data objects (protocols) may be sorted in such a tree.

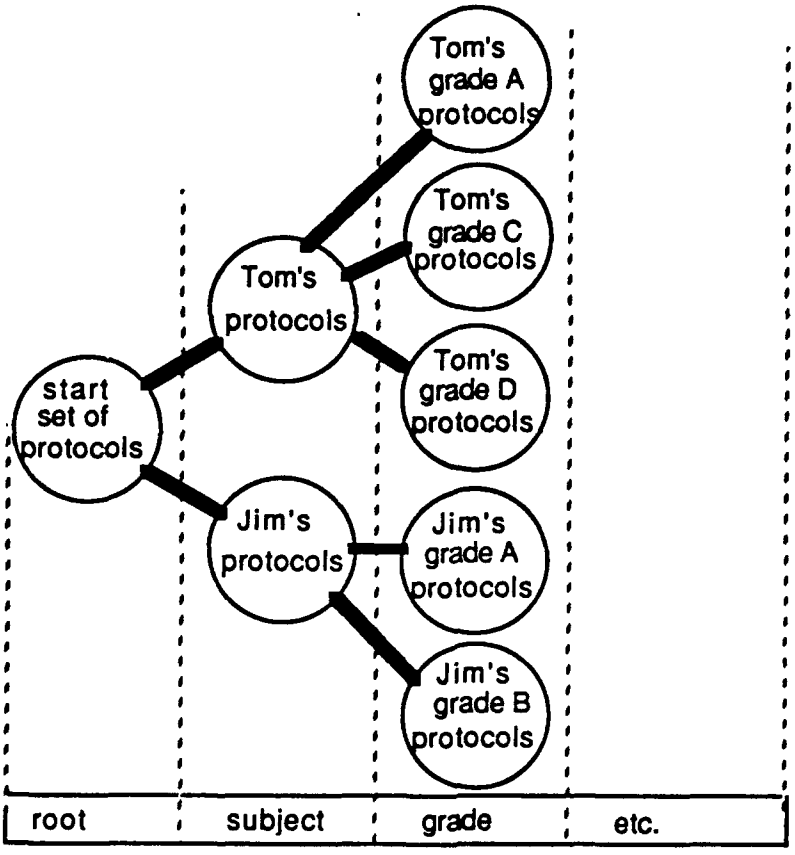


Figure 2:
AVtree for sorting protocols (data objects)

With this picture in mind, we move into the series of definitions concerning AVtrees:

Definition 1: Data Object

Let D be the set of all data objects.

Then $d \in D$ is a data object.

Data objects representing protocols are the original data objects in the system. The process of analysis generates further data objects. Thus, the intermediate results of analysis can be manipulated in a way similar to the original data sets. Any object identified by a set of attributes may be represented by a data object. No information is required about the contents of an object thus represented.

Definition 2: Attributes And Attribute-Value Space

Let A be the set of all *attributes*. Then $a \in A$ iff

$a = (f_a, V_a)$ where $nil \in V_a$ and

where f_a is a function such that

$f_a: D \rightarrow V_a$.

Then V_a is called the *value set* of the attribute. Instead of $f_a(d)$, we may write $a(d)$. (It is possible also to consider the set V, the union of the value sets over all attributes.)

For $a \in A$, (a,v) is an *AVpair* iff $v \in V_a$.

For each $a \in A$ and each $d \in D$, the position of d in the a dimension of the *attribute-value space* is $v = f_a(d)$.

Attributes and values represent the identifying information stored in the protocol's header. Attributes are defined as functions. If the protocol has a header with an attribute having a certain value (e.g.: subject = Tom), then the data object representing it will have a value defined at that attribute. Imagine data objects existing in a space with dimension equal to the cardinality of A, each dimension being labelled with an attribute of a. Then each object has some position in this space determined by its values on the attributes of A, where many of these values might be nil.

Definition 3: Attribute-Value Tree (AVtree)

For $a_1, \ldots, a_n \in A$, t is an *attribute-value tree* iff

$t = ((a_1, v_1), \ldots, (a_m, v_m), a_{m+1}, \ldots, a_n)$ for $1 \leq m \leq n$,

and $v_i \in V_{a_i}$ for $1 \leq i \leq m$

and if $i \neq k$, then $a_i \neq a_k$ for $1 \leq i \leq n$ and $1 \leq k \leq n$.

Notation:

root(t) = (a_m, v_m);

attr(t) = $(a_1, \ldots, a_n)$;

$attr_{sel}(t) = \{a_1, \ldots, a_m\}$ the set of select attributes of t;

$av_{sel}(t) = \{(a_1, v_1), \ldots, (a_m, v_m)\}$, the select AVpairs of t;

$attr_{sort}(t) = (a_m, \ldots, a_n)$ the sort attributes of t in order;

$vals(t) = \{v_1, \ldots, v_m\}$, the set of values of t;

height(t) = n - m.

The attribute-value ordered pairs (AVpairs) of an AVtree serve to select a set of data objects from the universe of all data objects, creating a starting set for the tree to sort. Each object in this subuniverse has the designated values on the selection attributes. The sorting attributes are then used to create a sorted tree from these objects. Notice that data objects are not mentioned in the definition. Their existence as contents of AVtrees is made explicit in the next definition.

Definition 4: Contents of an AVtree

For AVtree $t = ((a_1, v_1), \ldots, (a_m, v_m), a_{m+1}, \ldots, a_n)$,

the contents of t is

$c(t) = \{d \in D \mid a_i(d) = v_i \text{ for } 1 \leq i \leq m\}$.

t is an *empty* AVtree if $c(t) = \varnothing$.

Therefore, only the select attributes of t determine the contents of t, that is, the data objects it selects. To capture the idea of sorting the contents of an AVtree, consider each node of an AVtree also to be an AVtree. Then the contents of a subtree will have the characteristics of the selection AVpairs of that subtree.

Definition 5: Subtrees of an AVtree

For AVtree $t = ((a_1, v_1), \ldots, (a_m, v_m), a_{m+1}, \ldots, a_n)$, u is an *immediate subtree* of t ($u \in$ i-subtrees(t)) iff

$u = ((a_1, v_1), \ldots, (a_m, v_m), (a_{m+1}, v_{m+1}), a_{m+2}, \ldots, a_n)$,

and u is not empty.

Furthermore, s is a *subtree* of t ($s \in$ subtrees(t)) iff

$s \in$ i-subtrees(t)

or $s \in$ i-subtrees(r) where $r \in$ subtrees(t).

Notation for levels of subtrees:

$N^1(t)$ = i-subtrees(t);

$N^2(t)$ = i-subtrees(i-subtrees(t));

$N^k(t)$ = i-subtrees^k(t) for $1 \leq k \leq$ height(t).

A subtree specifies additional selection values in order to produce further subset divisions in the starting set of data objects. Some additional distinctions concerning these subtrees may be defined.

Definition 6: Special Types of AVtrees

t is a *terminal* AVtree iff t has only select attributes, that is, t is a list of AVpairs.

t is a *singleton* AVtree if c(t) contains only one data object.

Implications stated without proof:

1. A terminal AVtree has no subtrees.
2. The subtrees $N^{height(t)}(t)$ are terminal AVtrees known as terminals of the AVtree t (terminals(t)).
3. A singleton AVtree is not necessarily degenerate.
4. An AVtree with no subtrees is a terminal.

The terminals are like any other subtree except they are on the last level. Each terminal has as its contents a set of data objects, as do other subtrees. The terminals may be thought of as subtrees rooted at the leaves of t. Indeed, each subtree of t may be pictured as rooted at a node of t, the node being designated by an AVpair. Each subtree description subsumes the information contained in the description of its supertree, so the supertree can be derived from the subtree. The relation of the contents of the subtree and supertree remains undefined.

The next definition provides a rudimentary way of comparing AVtrees.

Definition 7: Coextensive AVtrees

Two AVtrees t and u are *coextensive* ($t \sim u$) iff

$c(t) = c(u)$.

Implication: For two AVtrees t and u, $t \sim u$ iff

$av_{sel}(t) = av_{sel}(u)$.

The forms of coextensive AVtrees may be completely different. All that is required is that they have the same set of select attributes in any permutation. This guarantees that the union of the contents of the terminals will contain the same set of data objects.

It is easy to describe the contents of AVtrees but to deal with their shapes is more difficult. In order to specify operators which produce results grouped according to the shape of the input AVtree, it is necessary to compare the shapes of trees.

Definition 8: Isomorphic AVtrees

Two AVtrees, t and u are *isomorphic* ($t \equiv u$) iff

$\exists\ \phi$:subtrees(t) $\rightarrow$ subtrees(u), a bijection such that for

$t_1, t_2 \in$ subtrees(t),

$t_2 \in N^1(t_1) \Leftrightarrow \phi(t_2) \in N^1(\phi(t_1))$.

Two AVtrees can be isomorphic even if they have completely different attributes. All that is required is that they branch at the same rate at their subtrees. The neck of selection AVpairs need not be the same length. Other than the neck, the two trees could be twisted around and laid on top of each other for a perfect match.

Another simple way of specifying the shape of an AVtree is the following:

Definition 9: Pruning

For two AVtrees t and u, u is a *k-pruning* of t ($u = \text{pruned}^k(t)$) iff

$t = ((a_1, v_1), \ldots, (a_m, v_m), a_{m+1}, \ldots, a_n)$,

$u = ((a_1, v_1), \ldots, (a_m, v_m), a_{m+1}, \ldots, a_q)$,

and

$k = n - q$ for $0 \leq k \leq n - m$.

Implication: All prunings of an AVtree are coextensive; that is, they have the same contents.

Pruning simply removes some number of bottom levels of an AVtree. Everything else stays the same, including the contents of the pruned tree.

Attribute-Value Operators

Having established the ability to group and manipulate data objects according to their values on attributes, the discussion turns to the problem of transforming groupings of these data objects. Operators will take an AVtree as input and produce an AVtree as output. An operator will need to know the general shape of input AVtree on which it can work, the types of data objects involved, and the type of output data objects it produces. A basic assumption of AVoperators is that their output tree is a pruning of the input tree. This constraint is meant to provide a graphical basis for associating groupings of results with groupings of input objects.

The framework for an operator consists of three pieces: a domain specification, a function on power sets of data objects, and the types of data objects it produces. The function associated with the operator produces groupings of new data depending on the groupings of the input

data objects. If each input data object produces an output object with the same attribute-values, then the output tree will be isomorphic with the input tree. If the function compares groups of data objects to produce one new group of data objects, the output objects will have the attributes common to all the input objects. Thus, they will form an AVtree one level shorter than the input tree. They represent a summary of the information in the input groupings. Such a function is illustrated in Figure 3.

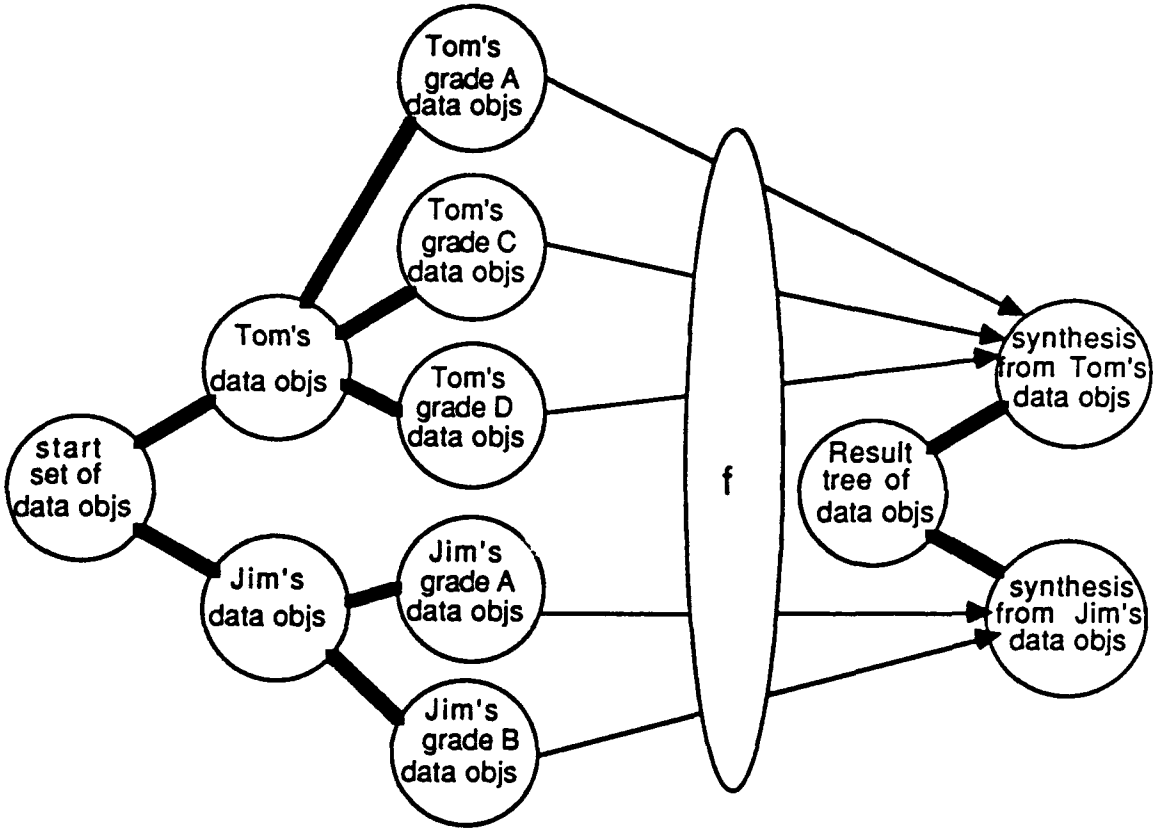


Figure 3:
First order synthesis function f
producing output AVtree from input AVtree

After defining AVoperators, we next describe how the operators may be chained to produce successively shorter AVtrees of data objects. This process draws to a summary point all the information present in the original AVtree. Any number of chains of operators may be applied to an input tree, each producing a different summary view of the input information. This is the goal of the protocol analysis system.

Definition 10: Multivalued Attributes

α is a *multivalued* attribute iff for $a_\alpha \in A$,

$\alpha = (a_\alpha, V_\alpha)$ where V_α is a cover of $V_{a\alpha}$.

$\alpha(d) = v_\alpha$ may be written when $a_\alpha(d) \in v_\alpha \in V_\alpha$.

Also, for $a \in A$, $a < \alpha$ (or a is a *subattribute* of α) iff

α is a multivalued attribute on a.

Furthermore, for (α, v_α) with $v_\alpha \subseteq V_\alpha$, a multivalued attribute and value set pair (*MAVpair*), and (a,v) an AVpair,

$(a,v) < (\alpha, v_\alpha)$ iff $a < \alpha$ and $v \in v_\alpha$.

If some variable α may be an attribute or multivalued attribute, $a <= \alpha$ may be written if $a = \alpha$ or $a < \alpha$ for attribute a.

A multivalued attribute specifies a covering set of subsets for the value space of some attribute. A multivalued attribute will be used below to specify the domain of an operator. This allows the operator to accept a range of values which fall into some specified set.

More difficult is to allow the operator to check the structure of the AVtree it is working on. The operator must know the general type of trees on which it can work. If it is to compare a to b and c to d and then compare those results, it has a subset structure like {{{a},{b}}, {{c},{d}}}.

The notation could quickly get out of hand without a more convenient form. The idea of subcontents captures the AVtree structure in a simple notation.

Definition 11: The Subcontents of AVtrees (see Figure 4)

Notation: For the set of all data objects D, the power set of D is represented by $\wp(D)$.

$\wp^2(D) = \wp(\wp(D))$; $\wp^3(D) = \wp(\wp^2(D))$.

Thus, if $x \in \wp^n(D)$, then $x \subseteq \wp^{n-1}(D)$.

Definition: The *subcontents* of t at some level is defined in terms of the contents of t, c(t), thus:

$c^0(t) = c(t)$;

$c^1(t) = \{B \in \wp(c(t)) \mid B = c(t') \text{ for } t' \in N^1(t)\}$;

Then,

$c^1(t) = \{c(t_{1,1}), c(t_{1,2}), \ldots c(t_{1,n})\}$;

$c^2(t) = \{c^1(t_{1,1}), c^1(t_{1,2}), \ldots c^1(t_{1,n})\}$;

$c^3(t) = \{c^2(t_{1,1}), c^2(t_{1,2}), \ldots c^2(t_{1,n})\}$;

$c^k(t) = \{c^{k-1}(t_1), c^{k-1}(t_2), \ldots c^{k-1}(t_n)\}$;

$c^{k+1}(t) = \{c^k(t_1), c^k(t_2), \ldots {}^k(t_m)\}$;

where $1 \leq k \leq \text{height}(t) - 1$.

Implication: $B \in c^k(t) \Rightarrow B \in \wp^k(c(t))$.

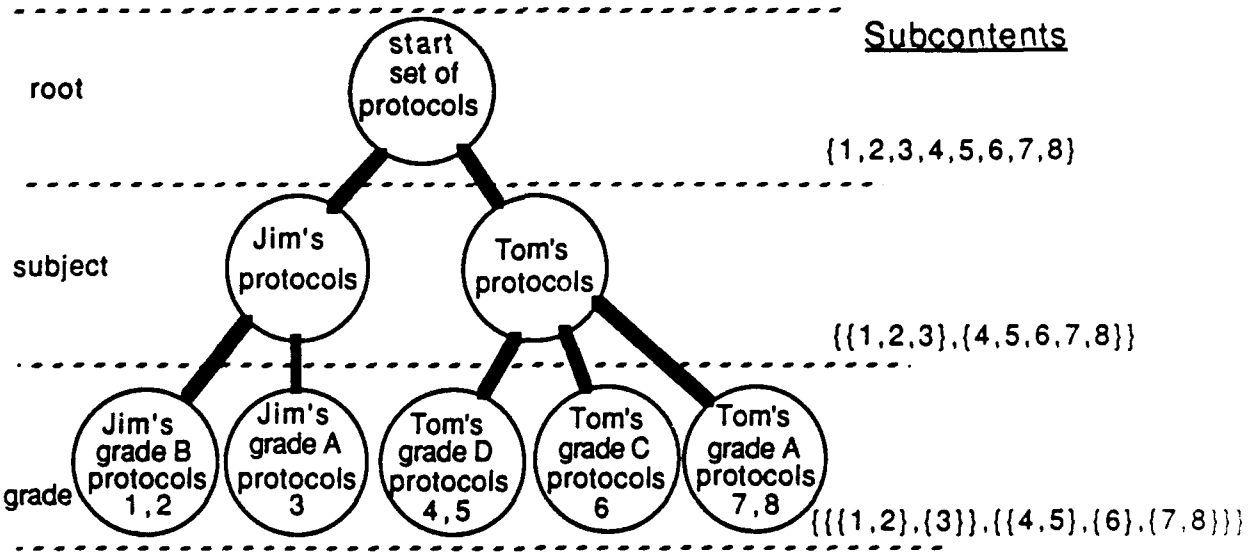


Figure 4:
Subcontents of an AVtree with 8 protocols

The subcontents notation provides an easy way to refer to the various levels of groupings applied to the contents of an AVtree. With these levels of groupings accessible, it becomes possible to consider a function which maps from some groupings of data objects as subcontents to an output data object. This function is peculiar in that it does not take a single object or vector of objects as input. Rather, it takes some order of power set of data objects. Such a function will be the heart of the AVoperator.

Definition 12: Synthesis Function

$S_0 = \{f \mid f: D \rightarrow D\}$,

$S_1 = \{f \mid f: \wp(D) \rightarrow D\}$,

$S_2 = \{f \mid f: \wp^2(D) \rightarrow D\}$,

and

$S_k = \{f \mid f: \wp^k(D) \rightarrow D\}$.

Then, if $f \in S_k$, f is a k-order *synthesis function*, and we write:

$order(f) = k$.

Implication: For k-order synthesis function f, and AVtree t with $k \leq height(t)$, $c^k(t)$ is in the domain of f.

Since a synthesis function maps elements of some order of power set of data objects to a single data object, it can be used to summarize those sets of data objects. In this way, it can collapse an AVtree into a shorter tree in order to create a more generalized representation of the input.

Such a function must be combined with two other pieces to form an AVoperator: a filter that selects trees of suitable shape and type; and a way to type the output of the function. With these three pieces known, AVoperators may be composed to produce new operators. The resulting AVtrees will form a regular pattern which can be easily drawn. Both the domain filter and the output form can be specified as sets of AVpairs. A set of AVpairs is the abstraction for a protocol header. The manipulation of these headers is, therefore, conveniently accomplished by AVoperators.

Definition 13: AVoperators

p is an *AVoperator* iff

$p = (t_p, f_p, r_p)$ where t_p is a set of AVpairs and/or MAVpairs, f_p is a synthesis function of order k, and r_p is a set of AVpairs unique for each output AVtree.

The definition begins by specifying the three parts of the AVoperator: the domain set, the synthesis function, and the result set. It then explains how the domain is tested and how the output data objects and AVtree will look. In particular, the output objects will be marked with uniquely identifying values in r_p: e.g., their creation operator, their input tree, and the time they were generated.

Definition 14: Domain of an AVoperator

For $t_p = \{(a_1, v_1), \ldots, (a_m, v_m)\}$,

then an AVtree u is in the domain of p ($u \in domain(p)$), iff

$order(f_p) \leq height(u)$,

and

$\forall\ (a_i, v_i) \in t_p$,

either $\exists\ (a', v') \in attr_{sel}(u) \mid (a', v') <= (a_i, v_i)$;

or $\exists\ a' \in attr_{sort}(u) \mid a' <= a_i$ and $v_i = V_{a'}$.

The domain set and the order of the synthesis function determine the input trees accepted. The domain set may include multivalued attributes. Thus, it can act as a range within which the input tree's AVpairs must fall. The input tree T is tested in two steps: First, its height must be sufficient for f_p to work on it. Second, all the AV/MAVpairs in the domain set must be matched or must contain an AVpair in T. A special case is the MAVpair whose value set is the value set for a sort attribute of T. This allows the operator to check only for the presence of an attribute in T and not for a particular value. If T passes both tests, the operator can produce from it an output tree which makes sense in the system.

Definition 15: Function of an AVoperator and Type of Its Result

For AVtrees T (input tree) and U (output tree),

with $T \in$ domain(p), and order(f_p) = k, then

p(T) = U, iff:

1. $av_{sel}(U) = \{(a_i,v_i) \mid (a_i,v_i) \in r_p \cup av_{sel}(attr_{sel}(T) - attr_{sel}(r_p))\}$ and $attr_{sort}(pruned^k(T)) = attr_{sort}(U)$.

2. $\forall\ \tau \in N^{height(U)}(T)$, where $f_p(c^k(\tau)) = d_\tau'$, then $\forall\ (a_i, v_i) \in av_{sel}(U)$, $a_i(d_\tau') = v_i$; and for $d_\tau \in c(\tau)$ and $a \in attr_{sort}(U)$, $a(d_\tau') = a(d_\tau)$.

U, the output AVtree, must be structured to uniquely select the created data objects and to sort them into a shape isomorphic with the input tree pruned. The objects must be typed to conform with this scheme. Since r_p uniquely marks the objects, it is sufficient for U to have these as selection AVpairs. In addition, U takes all AVpairs from T which do not conflict with r_p. Its sorting attributes are the same as T. The data objects are then made to agree with the select attributes of U and with the sort values held by their input data objects. Thus, they will be sorted into a tree isomorphic with T pruned according to the number of lower levels used up by f_p. The output objects will, therefore, be sorted to reflect the groupings of their source objects.

For practical purposes, a time stamp may be included in r_p since the contents of an AVtree may change. The same operator applied to the same AVtree may produce different results at different times depending on the universe of data objects. The AVtree is a filter for data and not a container. The time stamp allows the user to keep track of the origin of these different sets of results. The implementation, as discussed in the next section, must include a time-stamped database for exact recreation of an analysis session.

The three preceding definitions describe how to outline the structure of an AVoperator without detailing its specific action. Such an outlined operator can be tested and composed with other operators without having to worry about its actual function. Any operator, whether dummy or working, will pass on the information necessary to keep the relationships between output data objects consistent.

Since the chaining of operators produces narrower and narrower trees, it is possible to stack up the bottom layers of output trees into a pyramid which is shaped like the original input tree, though perhaps stretched out. This pyramid may be defined as the product of an input tree and a series of operators, producing a tree of stacked results.

Definition 16: Result Trees (Retrees)

For t, an AVtree, and $p_1 \ldots p_n$, AVoperators, r is a *Retree* iff

$r = (t, p_1, \ldots, p_n)$ where

$p_1(t) \in \text{domain}(p_2)$, and

$p_i(p_{i-1}(\ldots (p_1(t)) \ldots)) \in \text{domain}(p_{i+1})$, for $2 \leq i \leq n-1$.

Also, r is defined to be *fully specified* iff

$\text{height}(p_n(p_{n-1}(\ldots (p_1(t)) \ldots))) = 0$, and

$\text{contents}(p_n(p_{n-1}(\ldots (p_1(t)) \ldots))) = d$, a single data object.

r is *partially specified* if for t_n, the output of p_n,

$\text{height}(t_n) \neq 0$ and/or $|\text{contents}(t_n)| > 1$.

Implications:

1. For t_i, the output of p_i, t_i is isomorphic to a pruning of t shorter then the input tree of p_i by the order of f_{p_i}.
2. Any permutation of the sort attributes of the input tree of a Retree will still produce a Retree.
3. A new sort attribute inserted into the input tree of a Retree still produces a Retree.

The Retree represents a record of a sequence of operations concatenated with each other, starting on a particular input AVtree. Its structure reflects the structure of the input tree, relating result nodes to their various sources. The narrow top of the Retree contains those data objects which are produced from the previous levels of the tree. Therefore, the top of the Retree is a summary for what goes before. The Retree is subject to permutations determined by how the operators fit

together. It will be the basis for the visible representation of analysis results as described in the next section.

Definition 17: Operator Tree (Optree)

For the AVtree T,

optree(T) = {p is an AVoperator | T $\in$ domain(p)}.

Implication: For t $\in$ subtrees(T), optree(T) $\subseteq$ optree(t).

An Optree allows us to picture the composition of operators without considering functionality. The nodes (subtrees) of an AVtree may be thought of as containing sets of possible operators instead of data objects. For some AVoperator P and AVtree T, optree(P(T)) sorts out all the possible operators which may follow P applied to T.

The structures outlined above allow manipulation and transformation of protocols to be carried out according to regular patterns. They allow equivalencies and similarities to be spotted and exploited. Using this foundation, the relationship of various sortings and the interaction of different operators may be denotationally specified. The mathematics is not simple, but these definitions provide a means by which to develop an analysis system having pieces consistent with one another. They are the basis for the control of the prototype described in the next section.

Design for a Prototype

This section describes the design of a prototype protocol analysis system which is being implemented along the lines specified in the mathematical formalism. The system provides a graphical means for manipulating and operating on large numbers of machine-generated protocols. The principal graphical structure is a selection tree placed base-to-base with a tree of converging operations. This shape reflects the process of distinction followed by relation. The converging tree is formed by concatenating the increasingly narrow base layers of the output trees generated by succeeding operators. These layers combine to form the converging Result tree, a picture of the summary process (compare Figure 1, above).

The system consists of three modules: First, the *database* is an interface to the file system. It creates the universe of data objects from the files of protocols and serves as a communications link. The database also keeps a historical record of the objects in it at any particular time. Second, the *select-sort mode* is for creating and manipulating AVtrees. Third, the *operator mode* provides access to operators on AVtrees and access to the results of those operators.

Select-Sort Mode

The select-sort mode enables the visual selection and grouping of data objects as a necessary step in analysis. Data objects may be created from filed protocols or generated by operators to become data objects in the system. These results may also be selected and grouped for further analysis.

Figure 5 pictures an AVtree in select-sort mode. The select AVpairs form a neck stacked to the left. All data objects in the contents of the tree will satisfy these AVpairs. The tree is then sorted on the key attributes listed along the bottom menu bar. The data objects themselves ordinarily are not pictured but are grouped together in the terminal tree nodes. Here they are shown as the shaded leaf nodes, but they may be hidden by clicking a menu selection.

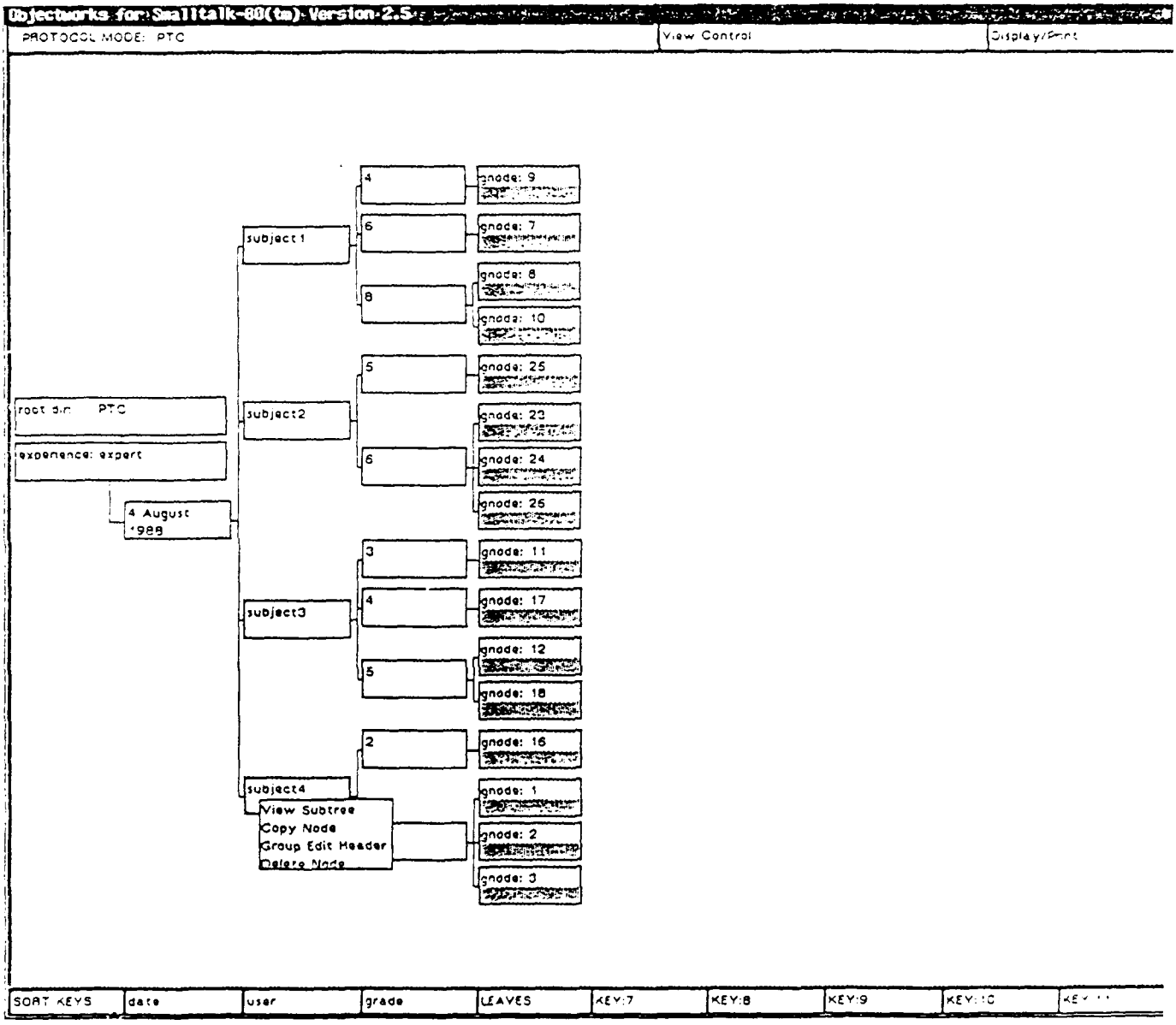


Figure 5:
Select-sort mode with AVtree grouping data objects

Many potentially complex tree manipulations are made simple. The sort attribute keys can be added, deleted, or shuffled, immediately giving a new sort tree. Any AVtree, defined as a series of AVpairs and attributes, can be saved as a filter and recalled later. Any supertree of the current tree can be reached by clicking on a node of the neck. Likewise, a subtree can be made the displayed tree by clicking on its root. Also the number of sublevels displayed for the current tree can be adjusted. These subtree changes and level selections combine to hide a great deal of information contained in a large set of data objects.

Direct access to the contents of the data objects is also provided through the select-sort mode. Changes to the AVpairs identifying each data object can be made on the local level or written back to the protocol files. Identical changes to all the contents of a particular subtree may be made with a single operation. An edit mode is also connected to the select-sort mode so that the protocol file associated with a data object can be edited.

Operator Mode

Figure 6 shows a Result tree which has been filled in by the operators listed along the bottom menu bar. Each operator produces a tree shaped like the input AVtree, only some number of levels shorter. In this case, each of the average operations produces a tree with one less level than the tree produced by the previous operator. When the bottom levels of these output trees are concatenated, the Result tree is formed.

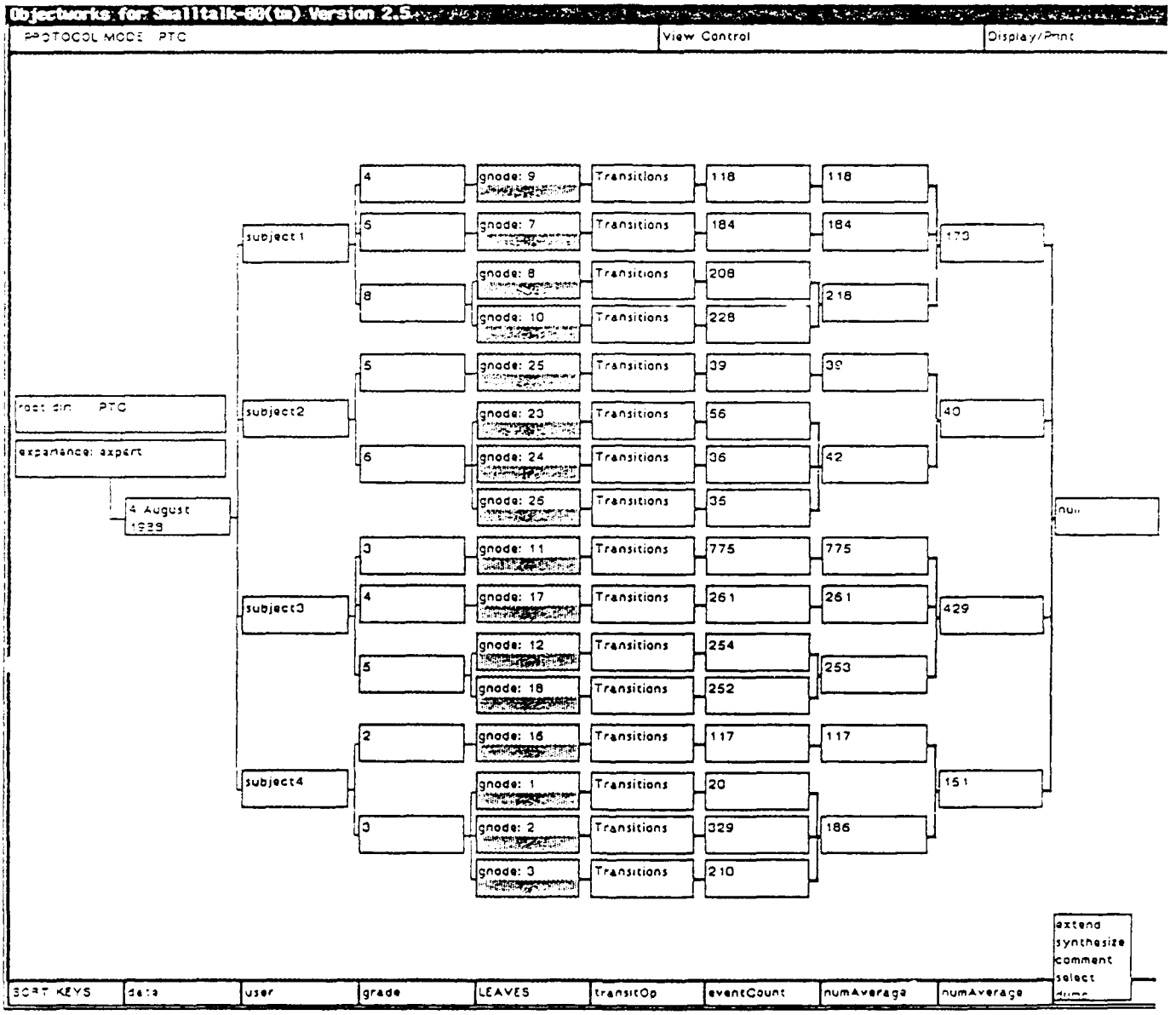


Figure 6:
Result tree produced from AVtree in operator mode

The first operator takes protocols as inputs. It reads the protocols and creates new protocols listing all events as generic transition events. The next operator counts the number of transitions in the new protocols. The third operator averages these counts. Because of the shape of the AVtree, these are averages for protocols grouped by subject and grade. The last operator also takes an average, this time for each subject. These last nodes represent a summary of all the input data objects in their peculiar groupings.

These operators are kept simple so as not to obscure the structure of the operator mode. In fact, the function of an operator may be complex, perhaps passing data to a statistical analysis package and receiving back a graphical output. Complex data objects are not viewed directly in operator mode, but rather in a data object viewing mode. All results are organized in the Result tree showing their relationships with each other and with the input data objects. Thus, relations between data objects are clarified by hiding much of their information, but that information is immediately available in a viewing mode.

The operations available at any point in the analysis are offered on a dynamic menu. The menu selections depend on the type and structure of the AVtree. Operators are queried as to whether they can work on the existing AVtree, and those which can are placed on the menu. The use of an operator is recorded in the bottom menu bar. A series of operators thus forms a script which can be reapplied to any suitable input tree. A script may be edited by moving and deleting operators, and the new Result tree is immediately produced. A script may also be composed into a single operator for repeated application and later use.

Operators do one of three things: produce a new attribute and value for a data object by looking at its data set; produce one new data object for one input object; or combine input data objects. The actual function of an operator can be programmed in the system or called externally, e.g., from a statistics package.

Conclusion

The massive and cryptic nature of machine-recorded protocols creates special difficulties for analysis. A tool for the manipulation of protocols and for the translation of their recorded languages into meaningful summaries ameliorates these difficulties. The mathematical specification described in this paper provides a logical and commodious framework within which such an analysis tool may be developed. The complexities due to massiveness and crypticness are hidden.

The analysis of protocols, however, has another problem which is common to all data analysis. Since it is exploration, data analysis is inherently difficult. Analysis has no path or algorithm to follow. It proceeds by experiment, experience, and insight. The mathematical model presented here constrains analysis by assuming a characteristic regularity. It supports a metamethod for analysis: the method of repeated distinction and relation. These restrictions provide a simplified problem more susceptible of solution.

A number of problems are not answered by this model. If the data sets are few in number or recorded without a regular notation, this model fails. Also, since operators may only be applied in sequence, only simple analysis sessions are repeatable. Finally, the connection of input objects to operators is strictly controlled by the form of the tree. One possible extension to this design would be a network mode for the linking of data and operations in a manner beyond the limitations of the two trees. A petri-net control on a network of data objects and operations would allow an operation to fire when presented with sufficient data of correct type. The groupings of data and the relation of output to input would be less clear in such a network mode. The difficulties of such a mode are similar to the general problem of programming-by-pictures. Another possible mode would be used to collect and compare similar analysis sessions, giving an overview of the variations among the sessions.

The use of computers in data analysis is an active field. The design of this protocol analysis system suggests two useful directions for further work. First, in analysis, temporary and adjustable constraints are an effective way to limit the otherwise overwhelming number of possibilities. Too many possibilities obscures potential solutions. Second, the construction of an operator should not require a complete algorithm. To be able to build approximate operators is useful if the nature of the result can only be partially specified. The ability to be

vague about what comes next is not natural to an algorithmic description but very natural for exploration.

The protocol analysis system modeled here seems to conform to some basic needs of the experimenter as well as offering feasible extensions. Within the framework of the mathematical description, a number of theorems might be developed to enable the automatic generation of AVtrees and application of operators for rapid testing of whole branches of the analysis space. Such potential power could prove useful for an experimenter. This model was originally designed to raise and explore such possibilities. The results suggest further effort in this direction might be fruitful.

Acknowledgments

This research was supported by NSF (Grants # IRI-8519517 and IRI-8817305), the Army Research Institute (Contract # MDA903-86-C-345) and by the Office of Naval Research (Contract # N00014-86-K-00680). Gordon Ferguson contributed to the content of this report, while John Smith contributed to both the content and the presentation.